

NPS55-80-023

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THE STAR FIELD MODULE

by

James K. ^{exr}HARTMAN
//

June 1980

Approved for public release; distribution unlimited.

Prepared for:

The U.S. Army Training & Doctrine Command
Fort Monroe, VA.

U
165
H3

U
165
H3

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

Jack R. Borsting
Provost

The work reported herein was accomplished with the support
of the U. S. Army Training & Doctrine Command, Fort Monroe, VA.

Reproduction of all or part of this report is authorized.

Prepared by:

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

I.	The Field Concept -----	1
A.	Introduction -----	1
B.	Fields -----	1
C.	Field Actions -----	3
D.	Overlapping Fields -----	5
II.	The Field Geometry Module for STAR -----	7
A.	Overview of the Module -----	7
B.	The Temporary Entity - FIELD -----	7
C.	The FLD.SET -----	9
D.	Routine FLD.CREATE -----	10
E.	Routine FLD.INIT -----	13
F.	Routine FLD.DIST -----	13
III.	Interfacing the Field Routines with STAR -----	18
A.	Overview of Changes Required -----	18
B.	PREAMBLE -----	18
C.	MAIN -----	19
D.	BL.CREATE and RED.CREATE -----	19
E.	MOVE -----	19
IV.	Field Actions -----	27
A.	Deciding Which Action to Perform -----	27
B.	Implementing Some Typical Actions -----	30

I. The Field Concept

A. Introduction

This report describes a portion of the STAR (Simulation of Tactical Alternative Responses) combined arms ground/air combat simulation model. STAR has been developed by students and faculty of the Operations Research Department at the Naval Postgraduate School as a tool for investigating ground/air combat. The model is programmed in SIMSCRIPT II.5, and a knowledge of that language is presumed in this report. Documentation of various other aspects of the model is given in references [1] - [5].

The purpose of the present report is to describe the Field Module of STAR. The Field Module is responsible for simulating vehicles encountering minefields and other similar battlefield features. The remainder of this chapter will introduce the general concept of a field and the two types of actions related to fields. Chapter II will describe the data structures representing fields and the basic subroutines for creating fields and monitoring vehicle location with respect to fields. Chapter III will discuss the modifications required in the existing STAR model to incorporate the Field Module. Finally, Chapter IV will consider the actions - the tactical responses - that occur as a result of fields.

B. Fields

Definition: A field is an area on the battlefield which influences battle actions and hence battle outcome.

Examples of fields include the following sorts of areas on the battlefield - some manmade and others naturally occurring:

Minefields - dug in or scatterable
Engineered or natural obstacles
River crossings
Bridges
Towns
Forests
Artillery trigger areas
Residual nuclear or chemical effects areas.

Forests are currently handled separately in the LOS module [2]. The Field Module will be set up to be able to handle the remaining fields.

Fields are represented in the STAR model as geometric areas overlaid on the terrain. The field module has the following functions:

1. For each element in the simulation, monitor when the element enters/exits each field.
2. When an element enters a field, perform designated actions (e.g. lower mine plow, reduce speed, change formation).
3. When an element leaves a field, restore its condition to the normal state.
4. Inside the field, perform designated actions (e.g. detonate mines).
5. Create fields at the beginning of the simulation and, if required, during the execution of the simulation.
6. Destroy fields during the simulation if required.

Since the Brigade level STAR model has the potential for modelling over a thousand elements, the field monitoring routines must be organized efficiently to avoid excessive overhead.

C. Field Actions

Field actions are things that happen because vehicles encounter fields. Typical actions which can be simulated include the following:

1. Change speed.
2. Change movement formation.
3. Stop for a given time, then continue.
4. Go into a hasty defensive posture.
5. Detonate a mine.
6. Lower or raise mine plow.
7. Call for artillery or air strike.
8. Mount or dismount infantry.
9. Change movement path in attempt to bypass field.
10. Attempt to clear obstacles or mines.
11. Attempt river crossing.
12. Respond to nuclear effects such as radiation, fires, blowdown.

The field actions which occur for a given field will typically depend on the field type (e.g. minefield), on parameters of the field (e.g. mine density), and on the tactical situation (e.g. under fire?). In this section we discuss the general concept of field actions and how they are triggered. Implementation of specific actions will be covered in Chapter IV.

There are two distinct kinds of field actions. Most field actions will occur immediately when vehicles cross the field boundaries (entering or exiting.) These actions, which we will call boundary actions, are easily triggered at the instant of entry or exit if the movement model can detect field boundary crossings. Other actions, however, occur somewhere inside the field. A mine detonation is the most obvious example. These actions are called internal actions. They are initiated by a field entry, but do not occur until the vehicle has moved into the field. To trigger

these actions, the simulation must measure the distance moved since field entry.

It should be noted that neither type of field action can be conveniently modelled by scheduling SIMSCRIPT events. This is because a scheduled event occurs after a designated time interval has passed, whereas field actions occur at designated locations. Since vehicles in STAR need not move at constant (or easily predictable) speed, it is usually impossible to predict when in time a field action will occur. Instead, the field module must monitor the location of each simulation element with respect to each field at every time of interest in the simulation. This is a task involving a potentially huge amount of computation.

One approach to monitoring location with respect to fields would use the movement model. For each element, at each move increment, the procedure would loop over all fields, testing whether the element was in the field. If the in/out status changed from a previous move increment, then a boundary action would be performed. This approach was considered and rejected for several reasons:

1. We suspect that it would be quite expensive to compute.
2. The approach as described only locates field boundaries to the accuracy of the current move increment's length.
(This could be overcome with additional computation).
3. Internal actions cannot be easily handled using this approach.

An alternative computational scheme is implemented in the current field module. For each element we compute and store the distance to the nearest field boundary action (entry, or exit) in the element attribute FLD.BDY.DIST(TANK). Similarly, we store the distance (if any) to a pending

internal action in the attribute FLD.INT,DIST(TANK). Then for each move increment these distances are decremented. When the remaining distance goes to zero, the appropriate action is triggered and the distance to the next action is computed. The main advantage of this process is that we need to loop over all fields only when a new distance is computed, and this will only be necessary when

1. The vehicle's direction of movement changes.
2. The vehicle encounters a field action.
3. New fields are created or old ones destroyed.

It also allows increased accuracy and handles boundary actions and internal actions using the same mechanism.

Details of the computation are reserved for Chapter II.

D. Overlapping Fields

A major problem which must be considered in planning the Field module is the problem of overlapping fields. If an element can be in several fields simultaneously, ambiguities can arise as to the intended field actions. The geometry of the field module can recognize multiple field entries and exits in whatever order they occur without any problems. However, making the action routines smart enough to take combinations of fields into account is probably prohibitively complex.

Unless special circumstances dictate otherwise, we plan to make the field action routines for a given field ignorant of whether the element is in other fields. Thus an element can safely be in two (or more) fields at one time as long as those fields influence different aspects of the element's behavior. If, however, field 1 increases an element's speed by 50% and field 2 decreases speed by 2 m./sec, then the effect of being in

both fields will be ambiguous. If the initial speed is 5 m/sec, then the resulting speed will be 5.5 m/sec if field 1 is entered first, or 4.5 m/sec if field 2 is entered first. The user should be aware of such possible ambiguities and avoid them by careful definition of the fields required for a given scenario.

Overlapping internal actions are also limited by the computational procedure of the Field module. Since an internal action is initiated (planned) when the field is entered, but doesn't actually occur until a designated distance into the field, we need to store this distance for each element (as an attribute called FLD.INT.DIST). Since only one such internal distance is stored for each element, the model will not correctly simulate an element which is simultaneously in two different fields which both involve internal actions.

These limitations are part of the current coding of the Field module. They doubtless could be overcome by more involved logic at an increased cost in coding effort, computer storage, execution time, and model clarity.

II. The Field Geometry Module for STAR

A. Overview of the Module

This chapter discusses the details of the geometric aspects of the field module. These include the definition and creation of fields and the process of monitoring field entry and exit. The Field module involves a new class of SIMSCRIPT temporary entities (FIELDS), along with several new subroutines for processing fields, which are discussed in this chapter. In addition, several existing routines must be modified to interface with the new module. (Chapter III).

B. The Temporary Entity - FIELD

In the STAR combat simulation, fields are modelled as elliptical overlays on the terrain. For purposes of field planning and model input data the shape of each field ellipse is described by the following parameters (see Fig. 1.):

XC.FLD } X and Y battlefield coordinates of the
YC.FLD } center of the ellipse

SMAJ.FLD semi-major axis length
SMIN.FLD semi-minor axis length

ANG.FLD orientation angle in radians measured
 counterclockwise from east to the major axis.

For computational purposes the boundary of the field is described by the quadratic equation

$$\begin{aligned} &PXX.FLD*(X-XC.FLD)**2 + PYY.FLD*(Y-YC.FLD)**2 \\ &+ PXY.FLD*(X-XC.FLD)*(Y-YC.FLD) = 1.0 \end{aligned}$$

Where the quadratic coefficients are defined as

$$\begin{aligned} CANG &= \cos(ANG.FLD) \\ SANG &= \sin(ANG.FLD) \\ PXX.FLD &= (CANG/SMAJ.FLD)**2 + (SANG/SMIN.FLD)**2 \\ PYY.FLD &= (SANG/SMAJ.FLD)**2 + (CANG/SMIN.FLD)**2 \\ PXY.FLD &= 2*SANG*CANG*(1.0/SMAJ.FLD**2 - 1.0/SMIN.FLD**2) \end{aligned}$$

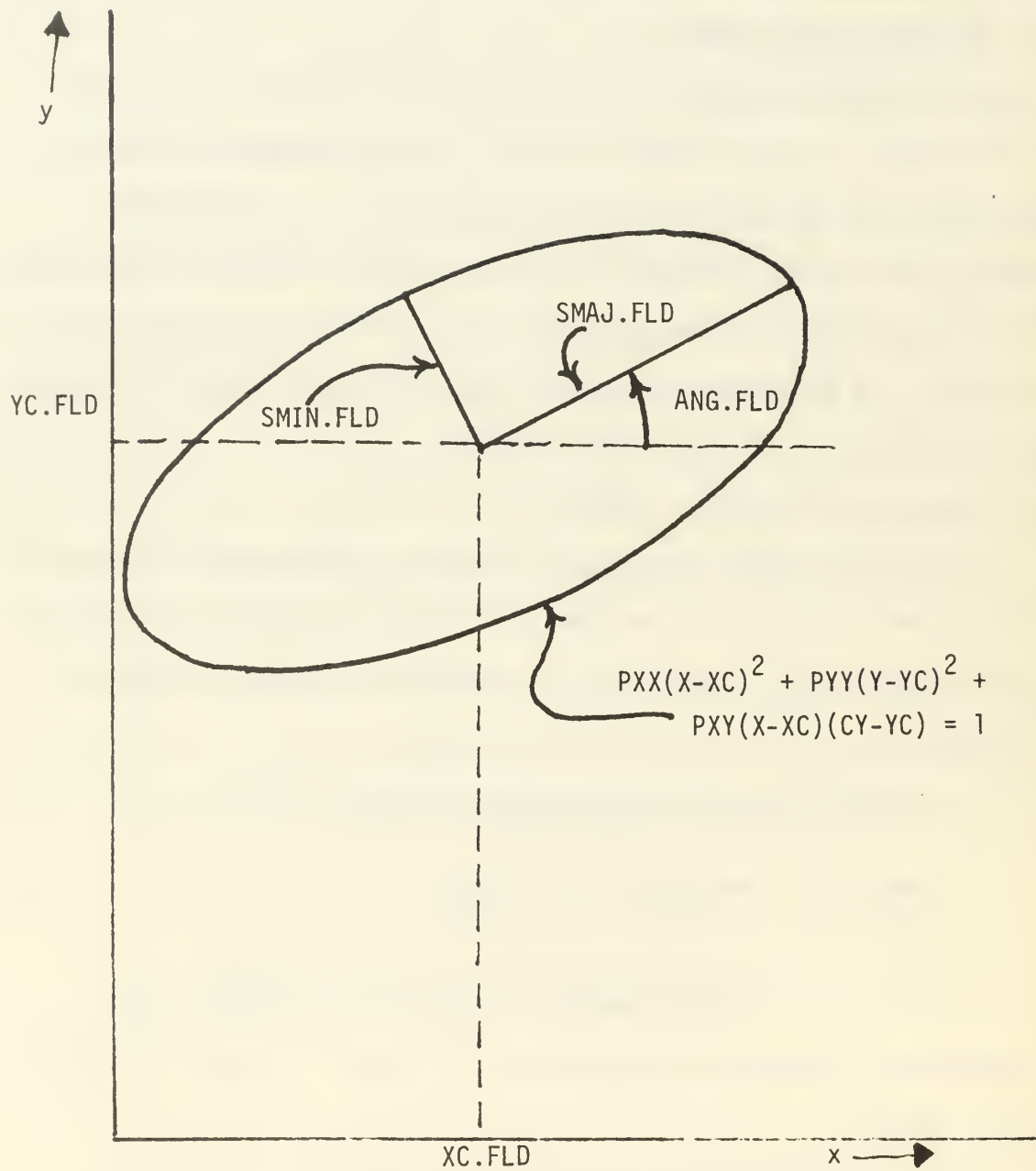


Figure 1. Field Shape Parameters

Each field is a SIMSCRIPT temporary entity in the entity class FIELD. The computational field parameters are stored as entity attributes as follows:

XC.FLD(FIELD)	}	Defined as above - all are real variables
YC.FLD(FIELD)		
PXX.FLD(FIELD)		
PYY.FLD(FIELD)		
PXY.FLD(FIELD)		

In addition, each field has several other attributes:

NAM.FLD	Sequential ID number in order of creation (Integer)	
TYP.FLD	Field type code (Integer)	
P1.FLD	}	Real variables defining other field parameters (e.g. minefield density) which influence field action routines. May have different meanings for different field types.
P2.FLD		
P3.FLD		
P4.FLD		

The distinction between the field type and the P1-P4 parameters is one of convenience. If two fields (say a scatterable minefield and a dug-in minefield) will both use the same field action routines, then they should be given the same field type, and the parameters should be used to distinguish between them. If, however, the actions triggered by one field are of a different sort than those resulting from the other field, then different field type codes should be used to describe them. We plan to have separate dedicated routines to trigger the actions for each separate field type. Each of these routines can use any or all of the P1-P4 parameters. Four such parameters are initially provided for; ultimately this number may change.

C. The FLD.SET

For convenience in looping over the currently active fields, a system-owned set called the FLD.SET has been declared. Each field entity is

filed in the FLD.SET as the field is created. For as long as the field remains in the FLD.SET, the simulation will consider it to be an active field and will process it in all field module computations. If a field is removed from the FLD.SET, then the field module will ignore it.

D. Routine FLD.CREATE

Field entities are created by the FLD.CREATE routine. In a typical STAR run, this routine will be called several times by the FLD.INIT routine before the start of the simulation to initialize all fields that are in place at the start of the battle. Then during the simulation, FLD.CREATE may be called at any time to create dynamically emplaced fields (e.g. artillery scatterable mines). The routine assigns the next unused sequence number to the NAM.FLD attribute of the new field and returns this value so that, if desired, the calling program can refer to this specific field in the future.

Given Arguments

XC	(real)	X coordinate of ellipse center
YC	(real)	Y coordinate of ellipse center
SAMAJ	(real)	semi-major axis length
SAMIN	(real)	semi-minor axis length
ANGLE	(real)	proper angle in radians from east to major axis
TYPE	(integer)	field type code
P1	(real)	} field parameters
P2	(real)	
P3	(real)	
P4	(real)	

Yielding Argument

NAME	(integer)	sequential ID number of this field
------	-----------	------------------------------------

Local Variables

SANG	(real)	sin(ANGLE)
CANG	(real)	cos(ANGLE)

Global Variables

FLDS.CREATED (integer) counter for number of fields created
so far

FIELD Entity Attributes

XC.FLD See section B of the chapter for
YC.FLD definitions - all are set by this
PXX.FLD routine for the newly created field.
PYY.FLD
PXY.FLD
NAM.FLD
TYP.FLD
P1.FLD
P2.FLD
P3.FLD
P4.FLD

Routines Called

None

Events Scheduled

None

Code See Figure 2.

Brief Description

Line 7 Creates the new field entity
Lines 8-14 Set field attributes directly from
the input arguments
Lines 15-17 Assign the sequence number
Lines 18-22 Compute the quadratic coefficients
Line 23 Files the field in the FLD.SET

Use Enter with descriptive geometric field parameters.
On exit the new field is created, and its computational
parameters are set.

```

1  ROUTINE FLD.CREATE
2  '' CREATE ONE FIELD AND SET ITS ATTRIBUTES
3      GIVEN XC,YC,SAMAJ,SAMIN,ANGLE,TYPE,P1,P2,P3,P4
4      YIELDING NAME
5      NORMALLY MODE IS REAL
6      DEFINE TYPE, NAME AS INTEGER VARIABLES
7      CREATE A FIELD
8      LET XC.FLD(FIELD) = XC
9      LET YC.FLD(FIELD) = YC
10     LET TYP.FLD(FIELD) = TYPE
11     LET P1.FLD(FIELD) = P1
12     LET P2.FLD(FIELD) = P2
13     LET P3.FLD(FIELD) = P3
14     LET P4.FLD(FIELD) = P4
15     ADD 1 TO FLDS.CREATED
16     LET NAM.FLD(FIELD) = FLDS.CREATED
17     LET NAME = FLDS.CREATED
18     LET SANG = SIN.F(ANGLE)
19     LET CANG = COS.F(ANGLE)
20     LET PXX.FLD(FIELD) = (CANG/SAMAJ)**2 + (SANG/SAMIN)**2
21     LET PYY.FLD(FIELD) = (SANG/SAMAJ)**2 + (CANG/SAMIN)**2
22     LET PXY.FLD(FIELD) = 2*SANG*CANG*(1/SAMAJ**2 - 1/SAMIN**2)
23     FILE THIS FIELD IN THE FLD.SET
24     RETURN
25 END

```

Figure 2. FLD.CREATE Routine

E. Routine FLD.INIT

The FLD.INIT routine is called once by MAIN before the start of simulation. It reads cards describing the fields to be created before the simulation starts and calls FLD.CREATE for each such field. The FLD.INIT routine is then released from MAIN. The code in Figure 3 is self explanatory with the possible exception of line 9 which converts the angle from degrees (on the input card) to radians for the simulation.

F. Routine FLD.DIST

The FLD.DIST routine computes the distance to the nearest field boundary along an element's direction of movement. The resulting distance is stored in the element's FLD.BDY.DIST attribute. The distance is computed by solving the quadratic equation for the intersection of the ellipse boundary with the movement ray as follows:

Let XCUR,YCUR be the current element location and DX,DY the components of a normalized direction vector for the element,

so $DX = \cos(\text{element angle of movement})$

and $DY = \sin(\text{element angle of movement})$.

Then $X = XCUR + S \cdot DX$ (1)

$Y = YCUR + S \cdot DY$

give a parametric form of the movement path where the parameter $S > 0$ represents distance moved.

The ellipse boundary is given by

$$P_{XX}(X-XC)^2 + P_{YY}(Y-YC)^2 + P_{XY}(X-XC)(Y-YC) = 1.0, \quad (2)$$

so plugging (1) into (2) gives a quadratic equation in S . If the quadratic has no real roots, then the movement path does not intersect the ellipse. If the roots are $S_1 = S_2$, then the solution is a tangency

```

1  ROUTINE FLD.INIT
2  ** CREATE ALL FIELDS THAT ARE IN PLACE AT THE START OF THE BATTLE
3  NORMALLY MODE IS REAL
4  DEFINE TYPE, NUM, I, NAME AS INTEGER VARIABLES
5  USE UNIT 5 FOR INPUT
6  READ NUM
7  FOR I = 1 TO NUM DO
8      READ XC,YC,SAMAJ,SAMIN,ANGLE,TYPE,P1,P2,P3,P4
9      LET ANGLE = ANGLE/RADIAN.C
10     CALL FLD.CREATE GIVEN XC,YC,SAMAJ,SAMIN,ANGLE,TYPE,P1,P2,P3,P4
11     YIELDING NAME
12 LOOP
13 RETURN
14 END

```

Figure 3. FLD,INIT Routine

and we ignore it. Otherwise both roots exist and $S1 < S2$. Negative roots correspond to boundaries already crossed, while positive roots correspond to boundaries which lie ahead. The routine loops over all fields and computes the smallest positive root.

Given Arguments

VEH	(Integer)	pointer to the element being moved
-----	-----------	------------------------------------

Local Variables

D	(double)	the smallest distance found so far
DX	(double)	cos of element movement direction
DY	(double)	sin of element movement direction
XCUR	(double)	} element's current location
YCUR	(double)	
PXX	(double)	} field boundary coefficients
PYY	(double)	
PXY	(double)	
A	(double)	} coefficients of the quadratic equation
B	(double)	
C	(double)	
		$AS^2 + BS + C = 0$
S1	(double)	} roots of the quadratic (if they exist)
S2	(double)	
		with $S1 < S2$

Element Entity Attributes

X.CURRENT	(real)	} element's current location and angle of movement
Y.CURRENT	(real)	
DIR.OF.MVMT	(real)	
FLD.BDY.DIST	(real)	result of the computation - distance to nearest boundary

Field Entity Attributes

XC.FLD	(real)	} field center coordinates
YC.FLD	(real)	
PXX.FLD	(real)	} field boundary coefficients
PYY.FLD	(real)	
PXY.FLD	(real)	

Routines Called

None

Events Scheduled

None

Code See Figure 4

Brief Explanation

Line 9 initializes D to + ∞

Lines 11-14 setup element location and direction

Lines 15-42 loop over each field in FLD.SET

Lines 16-30 compute S1, S2 if they exist, otherwise
 cycle to next field

Lines 31-38 find minimum positive value among S1, S2, D
 and save it in D

Line 47 stores the resulting D value (maybe + ∞)
 in the element's FLD.BDY.DIST attribute

Use The FLD.DIST routine is called for each element in the simulation when the element is created. Whenever an element changes movement direction or crosses a field boundary the routine is called for that element to give a new FLD.BDY.DIST. Note that the distance finally returned is limited to 500m. for reasons of numerical roundoff error. This forces the model to periodically re-evaluate the FLD.BDY.DIST attribute.


```

1  ROUTINE FLD.DIST
2  ** COMPUTE DISTANCE FROM VEH TO THE CLOSEST FIELD BOUNDARY
3  **  ALONG CURRENT DIRECTION OF MOVEMENT
4      GIVEN VEH
5  DEFINE DX,DY,XCUR,YCUR,QX,QY,PXX,PYY,PXY,A,B,C,ARG,SQ,S1,S2,D AS DOUBLE
6  VARIABLES
7      DEFINE VEH AS AN INTEGER VARIABLE
8  DEFINE FIELD AS AN INTEGER VARIABLE
9      LET D = RINF.C
10     IF FLDS.CREATED NE 0
11         LET DX = COS.F(DIR.OF.MVMT(VEH))
12         LET DY = SIN.F(DIR.OF.MVMT(VEH))
13         LET XCUR = X.CURRENT(VEH)
14         LET YCUR = Y.CURRENT(VEH)
15         FOR EVERY FIELD IN FLD.SET DO
16             LET QX = XCUR - XC.FLD(FIELD)
17             LET QY = YCUR - YC.FLD(FIELD)
18             LET PXX = PXX.FLD(FIELD)
19             LET PYY = PYY.FLD(FIELD)
20             LET PXI = PXI.FLD(FIELD)
21             LET A = PXX*DX**2 + PYY*DY**2 + PXI*DX*DY
22             LET B = 2*PXX*QX*DX + 2*PYY*QY*DY + PXI*(QX*DY+QY*DX)
23             LET C = PXX*QX**2 + PYY*QY**2 + PXI*QX*QY - 1.0
24             LET ARG = B**2 - 4.0*A*C
25             IF ARG LE 0
26                 CYCLE
27             ELSE
28                 LET SQ = SQRT.F(ARG)
29                 LET S1 = -(B + SQ)/(2.0*A)
30                 LET S2 = (SQ - B)/(2.0*A)
31                 IF S1 GT 0.0
32                     IF S1 LT D
33                         LET D = S1
34                     ALWAYS
35                 CYCLE
36             ELSE
37                 IF S2 GT 0.0 AND S2 LT D
38                     LET D = S2
39                 ALWAYS
40             **ENDIF
41         **ENDIF
42     LOOP
43     IF D LT RINF.C AND D GT 500.0
44         LET D = 500.0
45     ALWAYS
46     ALWAYS
47     LET FLD.BDY.DIST(VEH) = D
48     RETURN
49 END

```

Figure 4. FLD.DIST Routine

III. Interfacing The Field Routines with STAR

A. Overview of Changes Required

Several changes in the existing STAR model code are required to implement the field module. For the most part these changes are simple and fairly obvious, but in one case (the MOVE routine) substantial effort was required. The following existing program segments were changed:

PREAMBLE
MAIN
BL.CREATE
RED.CREATE
MOVE

We detail the required changes in the remaining sections of this chapter. Familiarity with the basic STAR ground model is assumed (refs [1]-[5]).

B. PREAMBLE

1. New global integer variable FLDS.CREATED
2. New system owned set FLD.SET
3. New class of temporary entities FIELD with attributes
NAM.FLD, XC.FLD, YC.FLD, PXX.FLD, PYY.FLD, PXY.FLD,
TYP.FLD, P1.FLD, P2.FLD, P3.FLD, P4.FLD
(see Chapter II for definition and type).
A FIELD may belong to the FLD.SET.
4. New attributes for the TANK (or UNIT) temporary entity:

FLD.INT.DIST	distance to pending internal action
FLD.BDY.DIST	distance to nearest field boundary
FLD.NO	name of the field involved in any pending internal action
FLD.AKT	code describing the kind of action for pending internal actions
5. FLD.INIT defined as a releasable routine.

C. MAIN

1. Immediately following the call to RES.TERR,

```
CALL    FLD.INIT          RELEASE FLD.INIT
```

to create fields.

D. BL.CREATE

1. Immediately after the call to INIT.POS,

```
CALL    FLD.DIST (TANK)
```

```
LET     FLD.INT.DIST(TANK) = RINF.C
```

to initialize field distance attributes.

E. RED.CREATE

1. Immediately after the call to INIT.POS,

```
CALL    FLD.DIST(TANK)
```

```
LET     FLD.INT.DIST(TANK)= RINF.C
```

to initialize field distance attributes.

F. MOVE

The changes to the MOVE routine are central to the Field module, and are the most critical. In this section we explain these changes in detail. To aid in understanding the relation of these changes to the rest of the MOVE routine, we include a listing of the entire routine in Figure 5, with changes flagged by "FLD in the right hand margin of the card. The reader is assumed to be familiar with the MOVE routine as documented in reference [4].

1. (Lines 157-161) Whenever a vehicle changes its direction of movement, the previously computed FLD.BDY.DIST is no longer valid, and must be recomputed by a call to FLD.DIST

2. (Lines 170-171) The distance limit for a move increment includes FLD.INT.DIST (to trigger internal actions) and FLD.BDY.DIST + 1 (to trigger boundary actions). The +1 is included to make sure that the boundary is actually crossed (by up to 1 meter) before the action is triggered. This prevents roundoff noise from triggering the same action more than one time.
3. (Lines 215-227) At the end of each movement increment, if there are any fields in the battle, FLD.BDY.DIST and FLD.INT.DIST are updated. If either distance hits zero, the FLD.ACT routine is called to decode and perform the action. If as a result of the field action, the vehicle can no longer move, the call to MOVE is terminated. Finally, if we are within 50 meters of a boundary, the FLD.DIST routine is called to get a more accurate FLD.BDY.DIST. This extra call is required because of round-off errors on the IBM-360 where vehicle X and Y coordinates have about 6 significant digits on a battlefield which may be 100,000 meters deep.

```

1  ROUTINE TO MOVE GIVEN VEH
2  DEFINE K AS AN INTEGER VARIABLE
3  DEFINE SLOPE AS A REAL VARIABLE
4  DEFINE REM.MOVE.TIME, DEL.X, DEL.Y, D.TO.MCP, ALPH, SALPH,
5      CALPH, GRAD.X, GRAD.Y, SPD.LIMIT, ACCEL.LIMIT, DECEL.LIMIT,
6      DIST.LIMIT, DEL.SPD, DIST.INCR, TIME.INCR AS REAL VARIABLES
7  DEFINE DIST.REQ, TIME.REQ, ZERO.LEVEL AS REAL VARIABLES
8  DEFINE X.DEST, Y.DEST, DIR, CX, CY, NX, NY, LX, LY, NLX, NLY, PX, PY,
9      NPX, NPY, X.OFF, Y.OFF, D.TO.DEST AS REAL VARIABLES
10 DEFINE THETA, CTH, STH AS REAL VARIABLES
11 DEFINE VEH, FINAL AS INTEGER VARIABLES
12 DEFINE MST, RT, NM, MCP.INC, LM, MCP, D.ON.RT AS INTEGER VARIABLES
13 DEFINE FAKE.MCP AS AN INTEGER VARIABLE
14 DEFINE I AND J AS INTEGER VARIABLES
15 CHECKOUT VEH ALWAYS
16 LET ZERO.LEVEL = 1.0      LET FINAL = 0
17 LET MST = MV.STATE(VEH)
18 IF MST = 0 OR MST >= 4 RETURN
19 ELSE
20 IF MST EQUALS 1
21     CALL RT.SEL(VEH)
22     LET MV.STATE(VEH) = 2
23 ALWAYS
24 LET REM.MOVE.TIME = TIME.V - T.SPD(VEH)
25 LET RT = ROUTE(VEH)
26 LET D.ON.RT = DIR.ON.RT(VEH)
27 LET FAKE.MCP = 0
28 IF RT EQUALS 0      LET D.TO.MCP = RINF.C      GO TO ANGLES
29 ELSE
30 ''CONSISTENCY CHECK FOR POSSIBLE TURNAROUND
31 IF AREA.START(VEH) LT AREA.END(VEH)
32     IF D.ON.RT EQ 0 GO TO NEW.MCP
33     ELSE LET D.ON.RT = 0
34     LET DIR.ON.RT(VEH) = 0
35     LET K = DIM.F(ROUTE.DATA(RT,*))/3
36     LET MCP = NEXT.MCP(VEH)
37     IF MCP EQ 0 LET NEXT.MCP(VEH) = 1
38     ELSE IF MCP EQ K LET NEXT.MCP(VEH) = 0
39     ELSE LET NEXT.MCP(VEH) = MCP + 1
40     ALWAYS
41 ALWAYS
42 ELSE ''AREA.START GT AREA.END
43     IF D.ON.RT EQ 1 GO TO NEW.MCP
44     ELSE LET D.ON.RT = 1
45     LET DIR.ON.RT(VEH) = 1
46     LET K = DIM.F(ROUTE.DATA(RT,*))/3
47     LET MCP = NEXT.MCP(VEH)
48     IF MCP EQ 0 LET NEXT.MCP(VEH) = K
49     ELSE IF MCP EQ 1 LET NEXT.MCP(VEH) = 0
50     ELSE LET NEXT.MCP(VEH) = MCP -1

```

Figure 5. MOVE Routine

```

51         ALWAYS
52     ALWAYS
53 ALWAYS
54 'NEW.MCP' LET MCP = NEXT.MCP(VEH)      LET NM = MCP * 3
55 IF MCP EQUALS 0  ''MOVE TO POSITION IN AREA.END
56     IF POS.IN.PLT.AREA(VEH) EQUALS 0, CALL BEST.POS(VEH)  ''SETTING POS.IN.PLT.A
57     ALWAYS
58     LET I = PLT(VEH)      LET K = POS.IN.PLT.AREA(VEH) * 3
59     FOR J = 1 TO DIM.F(POSITION(I,*,*)) WITH POSITION(I,J,1) EQUALS
60         AREA.END(VEH)
61     DO
62         LET X.DEST = POSITION(I,J,K-1)
63         LET Y.DEST = POSITION(I,J,K)
64         LET DIR = POSITION(I,J,K+1)
65     LOOP
66     LET D.TO.MCP = SQR.F((X.DEST-X.CURRENT(VEH))**2 +
67         (Y.DEST-Y.CURRENT(VEH))**2)
68     IF D.TO.MCP LESS THAN ZERO.LEVEL,
69         LET MV.STATE(VEH) = 4
70         LET DIR.OF.MVMT(VEH) = DIR
71         LET PRI.DIR(VEH) = DIR
72         LET SPD(VEH) = 0.
73         LET FINAL = 1
74         GO TO NEW.INCR
75     ELSE
76         GO TO DIRN.COMP
77 ELSE
78 IF FORM.CODE(VEH) EQUALS 0      ''GO DIRECTLY TO NEXT MCP
79     LET X.DEST = ROUTE.DATA(RT,NM-2)
80     LET Y.DEST = ROUTE.DATA(RT,NM-1)
81     LET D.TO.MCP = SQR.F((X.DEST-X.CURRENT(VEH))**2 +
82         (Y.DEST-Y.CURRENT(VEH))**2)
83     GO TO DIRN.COMP
84 ELSE      ''MOVE ALONG ROUTE OFFSET BY FORMATION
85 IF MCP EQUALS 1 AND D.ON.RT EQUALS 0  ''TOWARD FIRST MCP
86     LET NX = ROUTE.DATA(RT,4)
87     LET NY = ROUTE.DATA(RT,5)
88     LET LX = ROUTE.DATA(RT,1)
89     LET LY = ROUTE.DATA(RT,2)
90     LET I = ROUTE.DATA(RT,3)
91 ELSE
92     LET K = DIM.F(ROUTE.DATA(RT,*))
93     IF MCP EQUALS K/3 AND D.ON.RT EQUALS 1  ''TOWARD LAST MCP GOING BACKWARD
94         LET NX = ROUTE.DATA(RT,K-5)
95         LET NY = ROUTE.DATA(RT,K-4)
96         LET LX = ROUTE.DATA(RT,K-2)
97         LET LY = ROUTE.DATA(RT,K-1)
98         LET I = ROUTE.DATA(RT,K-3)
99     ELSE GO TO INTERMED
100 ALWAYS

```

Figure 5 (Continued).

```

101  ALWAYS
102      LET NLX = NX-LX      LET NLY = NY-LY
103      IF I EQUALS 0, LET I = FORM.CODE (VEH)
104      ALWAYS
105      LET J = FORM.POS (VEH) * 2
106      LET X.OFF = FORM.OFFSET (I,J-1)
107      LET Y.OFF = FORM.OFFSET (I,J)
108      LET THETA = ARCTAN.F (NLY,NLX)
109      LET CTH = COS.F (THETA)
110      LET STH = SIN.F (THETA)
111      LET X.DEST = LX + (X.OFF + FOR.CHG.INT)*CTH - Y.OFF*STH
112      LET Y.DEST = LY + (X.OFF + FOR.CHG.INT)*STH + Y.OFF * CTH
113      LET D.TO.MCP = SQRT.F ((X.DEST-X.CURRENT (VEH))**2 + (Y.DEST-Y.CURRENT (VEH))
114                      **2)
115      GO TO DIRN.COMP
116  'INTERMED'      '**TO HERE FOR INTERMEDIATE MCP'S ON ROUTE
117      LET CX = X.CURRENT (VEH)      LET CY = Y.CURRENT (VEH)
118      IF D.ON.RT EQUALS 0 LET LM = NM - 3
119      ELSE LET LM = NM + 3
120      ALWAYS
121      LET NX = ROUTE.DATA (RT,NM-2)
122      LET NY = ROUTE.DATA (RT,NM-1)
123      LET LX = ROUTE.DATA (RT,LM-2)
124      LET LY = ROUTE.DATA (RT,LM-1)
125      LET NLX = NX - LX      LET NLY = NY - LY
126      LET ALPH = -((CX-NX)*NLX + (CY-NY)*NLY) / (NLX*NLX + NLY*NLY)
127      LET PX = ALPH * LX + (1. - ALPH) * NX
128      LET PY = ALPH * LY + (1. - ALPH) * NY
129      LET NPX = NX - PX      LET NPY = NY - PY
130      LET I = ROUTE.DATA (RT,NM+3*(D.ON.RT-1))
131      IF I EQUALS 0, LET I = FORM.CODE (VEH)
132      ALWAYS
133      LET J = FORM.POS (VEH) * 2
134      LET X.OFF = FORM.OFFSET (I,J-1)
135      LET Y.OFF = FORM.OFFSET (I,J)
136      LET D.TO.MCP = SQRT.F (NPX*NPX + NPY*NPY)
137      IF D.TO.MCP LESS THAN ZERO.LEVEL GO TO MCP.REACHED
138      ELSE
139      LET THETA = ARCTAN.F (NLY,NLX)
140      LET CTH = COS.F (THETA)
141      LET STH = SIN.F (THETA)
142      LET ALPH = FOR.CHG.INT / D.TO.MCP
143      LET X.DEST = PX + ALPH * NPX + X.OFF * CTH - Y.OFF * STH
144      LET Y.DEST = PY + ALPH * NPY + Y.OFF * CTH + X.OFF * STH
145      LET D.TO.DEST = SQRT.F ((X.DEST-CX)**2 + (Y.DEST -CY)**2)
146      IF D.TO.DEST IS LESS THAN D.TO.MCP
147          LET D.TO.MCP = D.TO.DEST
148          LET FAKE.MCP = 1
149      ELSE LET FAKE.MCP = 0
150      ALWAYS

```

Figure 5 (Continued).


```

151  'DIAN.COMP'
152  IF D.TO.MCP IS LESS THAN ZERO.LEVEL,
153      GO TO MCP.REACHED
154  ELSE
155      LET DEL.X = X.DEST - X.CURRENT(VEH)
156      LET DEL.Y = Y.DEST - Y.CURRENT(VEH)
157      LET DIR = DIR.OF.MVMT(VEH)                                ''FLO
158      LET DIR.OF.MVMT(VEH) = ARCTAN.F(DEL.Y,DEL.X)
159      IF ABS.F(DIR-DIR.OF.MVMT(VEH)) GT 0.02                    ''FLO
160          CALL FLD.DIST(VEH)                                     ''TO COMPUTE NEW FLD.BDY.DIST  FLD
161      ALWAYS                                                    ''FLO
162      'ANGLES'
163      LET SALPH = SIN.F(DIR.OF.MVMT(VEH))      LET CALPH = COS.F(DIR.OF.MVMT(VEH))
164      'NEW.INCR' CALL ELEVG(X.CURRENT(VEH),Y.CURRENT(VEH)) YIELDING Z.CURRENT(VEH),
165      GRAD.X, GRAD.Y
166      IF FINAL EQUALS 1, GO TO END.MOVE
167      ELSE
168      LET SLOPE = GRAD.X * CALPH + GRAD.Y * SALPH
169      CALL MOVE.LIMITS GIVEN VEH, SLOPE YIELDING SPD.LIMIT, ACCEL.LIMIT, DECEL.LIMIT
170      LET DIST.LIMIT = MIN.F(D.TO.MCP,MAX.DIST.INCR,            ''FLO
171      FLD.INT.DIST(VEH), (1 + FLD.BDY.DIST(VEH)))                ''FLO
172      LET DEL.SPD= SPD.LIMIT - SPD(VEH)
173      IF ABS.F(DEL.SPD) LT 0.1,
174      ''EASY CASE -- NO ACCELERATION --
175          LET DIST.INCR = REM.MOVE.TIME * SPD.LIMIT
176          IF DIST.INCR IS GREATER THAN DIST.LIMIT,
177              ''MOVE STOPPED BY DISTANCE LIMIT
178              LET DIST.INCR = DIST.LIMIT
179              LET TIME.INCR = DIST.INCR / SPD.LIMIT
180      ELSE
181          ''MOVE STOPPED BY TIME LIMIT
182          LET TIME.INCR = REM.MOVE.TIME
183      ALWAYS GO TO MOVE.IT
184  ELSE ''HARD CASE -- ACCELERATION REQUIRED --
185      IF DEL.SPD IS LESS THAN 0, LET ACCEL.LIMIT = DECEL.LIMIT
186      ALWAYS LET TIME.REQ = DEL.SPD / ACCEL.LIMIT
187      LET DIST.REQ = SPD(VEH)*TIME.REQ + 0.5 * ACCEL.LIMIT * TIME.REQ **2
188      IF TIME.REQ IS GREATER THAN REM.MOVE.TIME,
189          ''SPD.LIMIT CANNOTBE ATTAINED IN REMAINING TIME, SO CHANGE LIMIT
190          LET SPD.LIMIT = SPD(VEH) + ACCEL.LIMIT * REM.MOVE.TIME
191          LET DIST.INCR = SPD(VEH) * REM.MOVE.TIME + 0.5 * ACCEL.LIMIT *
192          REM.MOVE.TIME ** 2
193      ELSE ''SPD.LIMIT CAN BE ATTAINED
194          LET DIST.INCR = DIST.REQ + (REM.MOVE.TIME - TIME.REQ)*SPD.LIMIT
195      ALWAYS
196      IF DIST.INCR IS LESS THAN DIST.LIMIT,
197          ''MOVE WILL BE STOPPED BY TIME LIMIT
198          LET TIME.INCR = REM.MOVE.TIME
199          LET SPD(VEH) = SPD.LIMIT
200      ELSE ''MOVE STOPPED BY DISTANCE LIMIT

```

Figure 5 (Continued).

```

201         LET DIST.INCR = DIST.LIMIT
202         IF DIST.LIMIT IS LESS THAN DIST.REQ,
203             LET TIME.INCR = (SQRT.F (SPD (VEH) **2+2.*ACCEL.LIMIT*DIST.LIMIT)
204                 -SPD (VEH))/ACCEL.LIMIT
205             ADD TIME.INCR * ACCEL.LIMIT TO SPD (VEH)
206         ELSE
207             LET TIME.INCR = TIME.REQ + (DIST.LIMIT-DIST.REQ)/SPD.LIMIT
208             LET SPD (VEH)=SPD.LIMIT
209         ALWAYS
210     ALWAYS
211     'MOVE.IT'     SUBTRACT TIME.INCR FROM REM.MOVE.TIME
212     ADD DIST.INCR * CALPH TO X.CURRENT (VEH)
213     ADD DIST.INCR * SALPH TO Y.CURRENT (VEH)
214     SUBTRACT DIST.INCR FROM D.TO.MCP
215     IF FLDS.CREATED GT D
216         SUBTRACT DIST.INCR FROM FLD.BDY.DIST (VEH)
217         SUBTRACT DIST.INCR FROM FLD.INT.DIST (VEH)
218         IF FLD.INT.DIST (VEH) LE 0.0 OR FLD.BDY.DIST (VEH) LE 0.0
219             CALL FLD.ACT (VEH)
220             IF KILL (VEH) EQ 1 OR MKILL (VEH) EQ 1
221                 LET FINAL = 1
222         ALWAYS
223     ALWAYS
224     IF FLD.BDY.DIST (VEH) LT 5D.D
225         CALL FLD.DIST (VEH)
226     ALWAYS
227     ALWAYS
228     IF REM.MOVE.TIME LT D.D1, LET FINAL=1
229     REGARDLESS
230     IF D.TO.MCP IS LESS THAN ZERO.LEVEL,
231     'MCP.REACHED'
232         IF FAKE.MCP EQUALS 1
233             LET FAKE.MCP = 0
234             GO TO NEW.MCP
235         ELSE
236             IF MCP = 0
237                 LET FINAL = 1
238                 GO TO NEW.MCP
239         ELSE
240             IF DIR.ON.RT (VEH) EQUALS D     '**MCP NUMBERS INCREASE
241                 IF NEXT.MCP (VEH) EQUALS DIM.F (ROUTE.DATA (ROUTE (VEH), *))/3
242                     LET NEXT.MCP (VEH) = 0
243                     GO TO NEW.MCP
244             ELSE
245                 ADD 1 TO NEXT.MCP (VEH)      GO TO NEW.MCP
246         ELSE     '**MCP NUMBERS DECREASE
247             IF NEXT.MCP (VEH) EQUALS 1
248                 LET NEXT.MCP (VEH)=0
249                 GO TO NEW.MCP
250     ELSE

```

Figure 5 (Continued).

```
251             SUBTRACT 1 FROM NEXT.MCP (VEH)      GO TO NEW.MCP
252 ELSE GO TO NEW.INCR
253 'END.MOVE'   LET T.SPD (VEH) = TIME.V
254 RETURN
255 END
```

Figure 5 (Continued).

IV. Field Actions

The final aspect of the Field module which we must discuss is the implementation of field actions. As indicated in Chapter I, field actions occur as a result of a vehicle encountering a field boundary. Actions which occur immediately are called field boundary actions. Actions which are planned for later occurrence are called field internal actions.

A. Deciding Which Action to Perform

When the FLD.ACT routine is called from the MOVE routine, it must decide which field action(s) to perform. Figure 6 shows a partial listing of a FLD.ACT routine. Details of particular actions are not included since they are quite dependent on the scenario plan for a given study. Examples of how typical actions might be handled are given in section B of this chapter. What this code segment does show is the procedure for deciding whether entry, internal, or exit actions should occur, and in each case, the field involved.

If FLD.INT.ACT(VEH) has been decremented to zero, then a previously planned internal action should occur, (lines 4-11). The field for which this action was planned is indicated in the vehicle's FLD.NO attribute and the action type is given by the vehicle's FLD.AKT attribute. Note that at most one such action can be pending at any one time. A typical internal action is a mine detonation which would call POP.A.MINE to assess lethality and, if the vehicle survived, would initiate evasive action and plan the next detonation for this vehicle.

If FLD.BDY.DIST(VEH) has been decremented to zero, then a boundary action should occur. A given move increment may, in some circumstances, cross more than one field boundary. Thus, instead of storing the field

```

1  ROUTINE FLD.ACT(VEH)  '' SORTS OUT WHICH FIELD ACTIONS TO PERFORM
2  DEFINE VEH, FIELD AS INTEGER VARIABLES
3  DEFINE DX,DY,XCUR,YCUR,QX,QY,PXX,PXY,PYY,A,B,C,ARG,SQ,S1,S2 AS DOUBLE VARIABLES
4  IF FLD.INT.DIST(VEH) LE 0.0
5      PRINT 1 LINE WITH NAME(VEH),X.CURRENT(VEH),Y.CURRENT(VEH),FLD.NO(VEH),
6          TIME.V AS FOLLOWS
7  FLD INT ACT  VEH=XXXX LOCN = XXXXXX XXXXXX FIELD = XXXX TIME = XXXX
8  ''
9  '' HERE WE CALL ROUTINES TO PERFORM INTERNAL ACTIONS
10 ''
11 ALWAYS
12 IF FLD.BDY.DIST(VEH) LE 0.0
13     LET DX = COS.F(DIR.OF.MVMT(VEH))
14     LET DY = SIN.F(DIR.OF.MVMT(VEH))
15     LET XCUR = X.CURRENT(VEH)
16     LET YCUR = Y.CURRENT(VEH)
17     FOR EVERY FIELD IN FLD.SET DO
18         LET QX = XCUR - XC.FLD(FIELD)
19         LET QY = YCUR - YC.FLD(FIELD)
20         LET PXX = PXX.FLD(FIELD)
21         LET PYY = PYY.FLD(FIELD)
22         LET PXY = PXY.FLD(FIELD)
23         LET A = PXX*DX**2 + PYY*DY**2 + PXY*DX*DY
24         LET B = 2*PXX*QX*DX + 2*PYY*QY*DY + PXY*(QX*DY+QY*DX)
25         LET C = PXX*QX**2 + PYY*QY**2 + PXY*QX*QY - 1.0
26         LET ARG = B**2 - 4.0*A*C
27         IF ARG LE 0
28             CYCLE
29         ELSE
30             LET SQ = SQRT.F(ARG)
31             LET S1 = -(B + SQ)/(2.0*A)
32             LET S2 = (SQ - B)/(2.0*A)
33             IF S1 LE 0.0 AND S1 GE -2.0
34                 PRINT 1 LINE WITH NAME(VEH),XCUR,YCUR,NAM.FLD(FIELD),
35                     TIME.V AS FOLLOWS
36 FIELD ENTRY  VEH=XXXX LOCN = XXXXXX XXXXXX FIELD = XXXX TIME = XXXX
37 ''
38 '' HERE WE CALL ROUTINES TO PERFORM FIELD ENTRY BOUNDARY ACTIONS
39 ''
40     ALWAYS
41     IF S2 LE 0.0 AND S2 GE -2.0
42         PRINT 1 LINE WITH NAME(VEH),XCUR,YCUR,NAM.FLD(FIELD),
43             TIME.V AS FOLLOWS
44 FIELD EXIT   VEH=XXXX LOCN = XXXXXX XXXXXX FIELD = XXXX TIME = XXXX
45 ''
46 '' HERE WE CALL ROUTINES TO PERFORM FIELD EXIT BOUNDARY ACTIONS
47 ''
48     IF NAM.FLD(FIELD) EQ FLD.NO(VEH)
49         LET FLD.INT.DIST(VEH) = RINF.C
50         LET FLD.NO(VEH) = 0

```

Figure 6. Partial FLD,ACT Routine

```
51             ALWAYS
52         ALWAYS
53     ''ENDIF
54     LOOP
55     ALWAYS
56     CALL FLD.DIST (VEH)
57     RETURN
58     END
```

Figure 6 (Continued).

number for a boundary action on the vehicle, we test each field for boundary crossing whenever FLD.ACT is called with $FLD.BDY.DIST \leq 0$. The loop to implement this test is contained in lines 17 to 54. Note the tolerance of ± 1 meter in lines 33 and 41. If this tolerance is decreased, some field entries or exits may be missed.

Regardless of the actions triggered in FLD.ACT, the FLD.BDY.DIST is recomputed prior to return to set up the next boundary crossing.

B. Implementing Some Typical Actions

In this section we provide suggestions for implementing some possible field actions. In many cases, an action performed on entering a field must be paired with a complementary action on exit which restores the vehicle to its previous state. This requires either saving the previous state for those attributes affected by the field or having a default background state to which exiting vehicles return.

The choice of field action to perform may, in some cases, depend on the type of "vehicle" encountering the field - an obstacle that totally stops wheeled vehicles may merely slow tanks and may be no hindrance at all for dismounted infantry. In this section we assume that the choice of actions to perform will be determined by the scenario writers and tacticians.

1. Change Speed

A frequent result of entering a field is a change in speed (e.g. for an obstacle field). In this section we discuss speed changes which do not completely stop the vehicle. Simply changing the vehicle's SPD attribute on field entry will NOT suffice, because this attribute is reset at every move increment based on the results of a call to routine MOVE.LIMITS. One way of implementing a speed change would be to define a new vehicle

attribute FLD.SPD.FAC the field speed factor for the vehicle. When the vehicle is not in any field, FLD.SPD.FAC assumes a background value of 1.0. On field entry the FLD.ACT routine would set FLD.SPD.FAC to a value greater than 1.0 for a speed increase and less than 1.0 for a speed decrease. (The value would be one of the field parameters P1 - P4). The MOVE.LIMITS routine would include FLD.SPD.FAC in its computation of vehicle target speed for each move increment. On field exit, FLD.ACT sets FLD.SPD.FAC back to 1.0.

2. Stop for T Seconds

To simulate a brief delay on field entry, the FLD.ACT routine can simply set the vehicle's MV.STATE to 3 (Stop along route). The vehicle will stop as soon as it is physically possible. For a longer delay, we probably also want to set the vehicle's DEFNUM to simulate taking advantage of available cover. In either case, once the vehicle is stopped it will not trigger further field action. To get the vehicle started again after T seconds (a field parameter), a new event must be scheduled by the field entry routine. No field action is required on field exit. Similar comments apply to the hasty defense action.

3. Response to Minefields

A variety of actions may be appropriate for simulating a vehicle encountering a minefield. The first question which must be asked is whether the vehicle notices the minefield: Several cases can occur:

- a) Minefield detected on entry - Then the field entry action routine can initiate any desired vehicle responses and will plan an internal mine detonation action.

- b) Minefield not detected - Then the field entry action routine only plans the internal mine detonation action, and the vehicle simply drives on. When the first detonation occurs, the minefield is assumed detected and appropriate responses are taken by the detonating vehicle and other vehicles in his immediate unit.
- c) Minefield detected prior to entry - A clearly visible field can be modelled by two concentric field ellipses. Entering the larger ellipse triggers the field detection and any responses thereto. Entering the smaller ellipse initiates mine detonation actions.

Given that a vehicle has detected the minefield, several actions may be appropriate for it and for other vehicles in its immediate unit (perhaps platoon). If mine plows or rollers are available, they may be used (vehicle PLOW.COND attribute). This probably requires a brief stop. Vehicles without plows may line up behind those with plows (a formation change). The resulting unit will move slower while plowing (speed change).

A more difficult action is an attempt to bypass the field. This would involve creation of special routes for moving around the field and has not been worked out in detail.

On field exit, if we assume that the vehicle(s) can detect the exit, the above actions can be cancelled (perhaps involving another brief delay). If the exit is not detectable, then concentric ellipses could be used to delay the change back to a normal condition until after the field has been left.

4. Mine Detonation

When a vehicle enters a minefield, the field boundary action is responsible for planning a mine detonation and storing this as the internal action for the vehicle. The distance that the vehicle will move before detonating a mine can be determined using a random draw from a distribution whose parameters are influenced by

- a) minefield density
- b) vehicle geometry (e.g. track width)
- c) mine detectability
- d) whether vehicle is in a cleared path
(e.g. following a vehicle with a plow)

This distance is used for the vehicle's FLD.INT.DIST attribute. If the vehicle travels this far through the field before encountering the exit boundary, then a field internal action is triggered. This action should assess the lethality of the mine, and, if the vehicle can still move, should plan the next detonation.

As discussed in 3) above, if the minefield is not detectable, then the first detonation in a platoon or company might trigger responses from all vehicles in that unit.

5. Call for Artillery or Air Strike

One possible use for field ellipses is to represent pre-planned artillery or close air support trigger areas. The presence of a threshold number of enemy vehicles in the area can be used to initiate requests for artillery or air missions. Implementing this concept using fields calls for establishing a counter for each such field to keep track of the number of enemy inside the field. On field entry, the counter is incremented, and perhaps a mission request is initiated. On field exit the counter is decremented.

Other possible uses for the Field module will no doubt become apparent in the future. Because of the great variety of possible combinations of the various actions, actual implementation of the actions must await guidance,

REFERENCES

- [1] HAGEWOOD, E. G. and WALLACE, W.S., Simulation of Tactical Alternative Responses (STAR), M.S. Thesis, Naval Postgraduate School, December 1978.
- [2] HARTMAN, J.K., "Parametric Terrain and Line of Sight Modelling in the STAR Combat Model", Naval Postgraduate School, Technical Report NPS55-79-018, August 1979.
- [3] CALDWELL, W.J. and MEIERS, W.D., An Air to Ground and Ground to Air Combined Arms Combat Simulation (STAR-AIR), M.S. Thesis, Naval Postgraduate School, September 1979.
- [4] PARRY, S.H. and KELLEHER, E.P. Jr., "Tactical Parameters and Input Requirements for the Ground Component of the STAR Combat Model," Naval Postgraduate School, Technical Report NPS55-79-023, October 1979.
- [5] HARTMAN, J. K., "Ground Movement Modelling in the STAR Combat Model," Naval Postgraduate School, Technical Report NPS 55-80-021, May 1980.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
4. Professor James K. Hartman, Code 55Hh Department of Operations Research Naval Postgraduate School Monterey, California 93940	40
5. Professor S. H. Parry, Code 55Py Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
6. LTC Edward P. Kelleher, Code 55Ka Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
7. Professor Arthur L. Schoenstadt, Code 53Zh Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
8. Office of the Commanding General U.S. Army TRADOC ATTN: General Donn A. Starry Ft. Monroe, Virginia 23651	1
9. Headquarters U.S. Army TRADOC ATTN: ATCG-T (Colonel Ed Scribner) Ft. Monroe, Virginia 23651	1
10. Headquarters U.S. Army TRADOC ATTN: Director, Analysis Directorate Combat Developments Ft. Monroe, Virginia 23651	1
11. Headquarters U.S. Army TRADOC ATTN: Director, Maneuver Directorate Combat Developments (COL Fred Franks) Ft. Monroe, Virginia 23651	1

12. Mr. David Hardison 1
Deputy Under Secretary of the Army
(Operations Research)
Department of the Army, The Pentagon
Washington, D.C. 20310
13. LTG William Richardson 1
Commanding General
U.S. Army Combined Arms Center
Ft. Leavenworth, Kansas 66027
14. Director 1
Combined Arms Combat Development Activity
ATTN: COL Reed
Ft. Leavenworth, Kansas 66027
15. Director, BSSD 1
Combined Arms Training Development Activity
ATTN: ATZLCA-DS
Fort Leavenworth, Kansas 66027
16. Director 1
Combat Analysis Office
ATTN: Mr. Kent Pickett
U.S. Army Combined Arms Center
Fort Leavenworth, Kansas 66027
17. Command and General Staff College 1
ATTN: Education Advisor
Room 123, Bell Hall
Fort Leavenworth, Kansas 66027
18. Dr. Wilbur Payne 1
Director
U.S. Army TRADOC Systems Analysis Activity
White Sands Missile Range, New Mexico 88002
19. Headquarters 1
Department of the Army
Office of the Deputy Chief of Staff
for Operations and Plans
ATTN: LTG Glenn Otis
Washington, D.C. 20310
20. Commander 1
U.S. Army Concepts Analysis Agency
8120 Woodmont Avenue
ATTN: MOCA-SMS (CPT Steve Shupack)
Bethesda, Maryland 20014
21. Commander 1
U.S. Army Concepts Analysis Agency
ATTN: LTC Earl Darden-MOCA-WG
8120 Woodmont Avenue
Bethesda, Maryland 20014

22. Director 1
U.S. Army Night Vision & Electro-optical Lab.
ATTN: DEL-NV-V1 (Mr. Bob Hermes)
Fort Belvoir, VA 22060
23. Director 1
U.S. Army Material Systems Analysis Activity
ATTN: Mr. Will Brooks
Aberdeen Proving Grounds, Maryland 21005
24. Director 1
Armored Combat Vehicle Technology Program
ATTN: COL Fitzmorris
U.S. Army Armor Center
Fort Knox, Kentucky 40121
25. Colonel Frank Day 1
TRADOC Systems Manager-XM1
U.S. Army Armor Center
Fort Knox, Kentucky 40121
26. Director 1
Combat Developments, Studies Division
ATTN: MAJ W. Scott Wallace
U.S. Army Armor Agency
Fort Knox, KY 40121
27. Commandant 1
U.S. Army Field Artillery School
ATTN: ATSF-MBT (CPT Steve Starner)
Fort Sill, Oklahoma 73503
28. Director 1
Combat Developments
ATTN: COL Clark Burnett
U.S. Army Aviation Agency
Fort Rucker, Alabama 36360
29. Director 1
Combat Developments
U.S. Army Infantry Agency
Fort Benning, GA 31905
30. Director 1
Missile Intelligence Agency
ATTN: ADA Tactics (CPT E.G. Hagewood)
Redstone Arsenal, AL 35809
31. Director 1
Combat Developments
ATTN: CPT William D. Meiers
U.S. Army Air Defense Agency
Fort Bliss, TX 79905
32. Commander 1
U.S. Army Logistics Center
ATTN: ATCL-OS-Mr. Cammeron/CPT Schuessler
Fort Lee, VA 23801

33. Commander 1
USAMMCS
ATTN: ATSK-CD-CS-Mr. Lee/Mr. Marmon
Redstone Arsenal, AL 35809
34. Commander 1
U.S. Army Combined Arms Center
ATTN: ATZL-CA-CAT (R.E. DeKinder, Jr.)
Fort Leavenworth, KA 66027
35. Director 1
U.S. Army AMSAA
ATTN: DRXS-YY-AA (Mr. Tom Coyle)
Aberdeen Proving Grounds, MD 21005
36. Chief 10
TRADOC Research Element Monterey (TREM)
Naval Postgraduate School
Monterey, CA 93940
37. Office of the Deputy Chief of Staff 1
for Combat Developments
U.S. Army TRADOC
ATTN: Major General Carl Vuono
Fort Monroe, Virginia 23651
38. Deputy Commanding General 1
Combined Arms Combat Development Activity
ATTN: ATZL-CA-DC (BG(P) Jack Walker)
Fort Leavenworth, KA 66027
39. Commanding General 1
U.S. Army Infantry Center
ATTN: Major General David Grange
Fort Benning, GA 31905
40. Director 1
Combat Developments
ATTN: COL Pokorney
U.S. Army Field Artillery Center
Fort Sill, Oklahoma 73503
41. Director 1
Combat Developments
ATTN: COL Gus Watt
U.S. Army Infantry School
Fort Benning, GA 31905
42. Director 1
USATRASANA
ATTN: Mr. Ray Heath
White Sands Missile Range, New Mexico 88002
43. R. Stampfel, Code 55 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

U
165 (Dupl. Card) 188272 2
H3 Hartman
The STAR field module.

6 AUG 82

16 NOV 82

5 OCT 86

26 JAN 88

27024

29152

31601

33563

U
165 Hartman 188272
H3 The STAR field module.

genU 165.H3

The STAR field module /



3 2768 000 43409 6

DUDLEY KNOX LIBRARY